Computing with radial basic functions the Beatson-Light way!

Will Light

Department of Mathematics and Computer Science, University of Leicester, UK.
pwl@mcs.le.ac.uk

Abstract

In this paper we discuss a number of recent developments in the practice of how to compute with radial basic functions. The two main problems addressed are how to develop fast evaluation schemes for radial basic functions, and how to efficiently carry out the solution of the interpolation problem. The approach is to mainly describe work which has involved the author and Professor Rick Beatson as contributors, and to include an idiosyncratic selection of works by other researchers which have attracted the attention of the author.

1 Introduction

Research into radial basic functions has been active now for about 30 years. The basic setup is as follows. A function $\psi : \mathbb{R}^n \to \mathbb{R}$, which we refer to as the *basic function*, is specified. A subspace is then constructed by reference to points $x_1, \ldots, x_m$ in $\mathbb{R}^n$. The members of this subspace all have the form

$$s(x) = \sum_{i=1}^{m} a_i \psi(x - x_i), \qquad x \in \mathbb{R}^n,$$

where the $a_1, \ldots, a_m$ are real numbers. It is important to appreciate at the outset that throughout this paper, and indeed in most of the papers appearing in this area, the underlying assumption is that the points $x_1, \ldots, x_m$ are distinct. One of the most common tasks for which these functions are used is interpolation. A small amount of research has been carried out where the points at which an interpolant is developed are arbitrary distinct points in $\mathbb{R}^n$, but by far the majority of the work relates to interpolation which is carried out at the same points as those used to effect the translation. Accordingly, data $d_1, \ldots, d_m$ are given at $x_1, \ldots, x_m$, and we require that

$$d_j = s(x_j) = \sum_{i=1}^{m} a_i \psi(x_j - x_i), \qquad j = 1, \ldots, m. \tag{1.1}$$

Two immediate observations present themselves. Firstly, at the present level of generality there is absolutely no guarantee that the Equations (1.1) will have a unique solution. Secondly, one knows from the work of Mairhuber [14] that there are no Haar subspaces

of significant dimension in any space $\mathbb{R}^n$ for $n \geq 2$. What this means is that if we are to construct interpolation problems which have a unique solution for each location of the data points $x_1, \ldots, x_m$ and for each choice of the data $d_1, \ldots, d_m$, then the subspace used must vary as the interpolation points vary. If we pause for a moment and consider how we might in some sensible and orderly way vary the subspace as the points $x_1, \ldots, x_m$ vary, then using simple shifts of a single basic function ψ is one of the most natural choices. It is very common to work with a function ψ which is a radial function. Thus we take a function $\phi : \mathbb{R}^+ \to \mathbb{R}$ and determine ψ by the rule $\psi(x) = \phi(|x|)$ for all $x \in \mathbb{R}^n$. Note that throughout this account, the symbol $|\cdot|$ will stand for the Euclidean norm in $\mathbb{R}^n$. At this point a common inaccuracy arises. The function ψ can be correctly referred to as a *radial basic function.* However, many authors give this appellation to the function ϕ, whose radiality is of no consequence whatsoever, since it would imply that ϕ was simply an even function on $\mathbb{R}$. Since ϕ only acts on $\mathbb{R}^+$ the idea that ϕ can be radial is vacuous. Let us continue in this spirit of criticism a little while longer. As far as the author is aware, only two people in the world would refer to ψ as a basic function, or a radial basic function. All other authors would use the word *basis* in place of *basic.* There are very obvious problems with this terminology. We are seeking to generate subspaces which are suitable for interpolation. Such subspaces will naturally have the same dimension as the number of data, and the functions $\{\psi(\cdot - x_i) : i = 1, \ldots, m\}$ should form a basis for the subspace. The use of the word basis in two completely different senses seems to the author to be misleading and unhelpful, whereas use of the word basic — a difference of one character — eliminates any possibility of confusion, and avoids the use of the word *basis*, which has a very specific mathematical meaning, in a context where its meaning is not the usual mathematical one.

The problem about whether interpolation is possible has a highly satisfactory answer in the work of Micchelli [15]. We direct the reader to the book of Cheney and Light [10] for a full account of these matters. A couple of examples will be helpful. If one chooses

$$s(x) = \sum_{i=1}^{m} a_i \phi(|x - x_i|) = \sum_{i=1}^{m} a_i \exp(-|x - x_i|^2), \qquad x \in \mathbb{R}^n,$$

or

$$s(x) = \sum_{i=1}^{m} a_i \phi(|x - x_i|) = \sum_{i=1}^{m} a_i |x - x_i|, \qquad x \in \mathbb{R}^n,$$

then the resulting interpolation problem is uniquely solvable for any choice of $x_1, \ldots, x_m$ and for any data $d_1, \ldots, d_m$. This result contrasts very strongly with the case for polynomial interpolation, where the data points $x_1, \ldots, x_m$ have to be constrained not to lie on an algebraic surface of appropriate degree. Indeed, the alternative formulation of the above result for the second example is quite often surprising to mathematicians who are uninitiated in the theory of radial basic functions.

Theorem 1.1 *Let $x_1, \ldots, x_m$ be distinct points in $\mathbb{R}^n$. Then the matrix $(|x_j - x_i|)$ is invertible.*

Having drawn a clear distinction between polynomial approximation and approximation by (radial) basic functions it is at this point that we must consider having some

polynomial ingredients in our interpolant. This is done in a very standard way by a process we call augmentation by polynomials. We consider interpolants of the form

$$s(x) = \sum_{i=1}^{m} a_i \phi(|x - x_i|) + p(x), \qquad (x \in \mathbb{R}^n).$$

Here p is a polynomial of total degree at most $k-1$. We still wish to interpolate to m pieces of information, but now have more than m parameters to determine with this information. The remaining parameters are determined via the 'natural' boundary conditions. The full set of equations is

$$\begin{aligned} d_j &= s(x_j) = \sum_{i=1}^{m} a_i \phi(|x_j - x_i|), \qquad j = 1, \ldots, m \\ 0 &= \sum_{i=1}^{m} a_i q(x_i), \qquad \text{for all } q \in \pi_{k-1}(\mathbb{R}^n). \end{aligned}$$

Here $\pi_{k-1}(\mathbb{R}^n)$ represents the space of polynomials of total degree $k-1$ in $\mathbb{R}^n$. Two questions present themselves pretty quickly from this additional hypothesis. Why should polynomials be added to the interpolant, and why are the boundary conditions chosen in this particular way? In some sense it is essential that we allow ourselves the possibility of adding polynomial terms to some of the interpolants, as we shall soon see. The most important example of a radial basic function interpolant which has a polynomial part will be the thin-plate spline. We will make considerable reference to this interpolant in $\mathbb{R}^2$, where it has the form

$$s(x) = \sum_{i=1}^{m} a_i |x - x_i|^2 \ln |x - x_i| + ax + b, \qquad (x \in \mathbb{R}^2).$$

Note here that the parameter a is a vector with two entries, as is x. Thus ax stands for the dot product between a and x. The parameter b is a real number. The natural boundary conditions take the form

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} a_i s_i = \sum_{i=1}^{m} a_i t_i = 0,$$

where $x_i = (s_i, t_i)$, $i = 1, \ldots, m$. This particular interpolant exhibits a feature common to all the cases where augmentation by polynomials is either necessary or desirable: the degree of the polynomial added is very low. The usual choices are $k = 0$ (when no polynomial term is added), $k = 1$ (when the term is a constant polynomial) and $k = 2$ (when the added polynomial is linear). It is now no longer possible to carry out interpolation for all choices of the points $x_1, \ldots, x_m$. One must avoid distributions of these points which lie on a zero surface of the corresponding polynomial subspace. In the explicit case we considered above (thin-plate splines), the very mild restriction needed is that $x_1, \ldots, x_m$ should not all lie on a single straight line. The theory developed by Micchelli [15] includes the case of augmentation by polynomials.

We now propose to take a look at a very simple example which we hope will give the

reader a feel for some of the ideas and concepts we have introduced so far. We consider

$$s(x) = \sum_{i=1}^{m} a_i|x - x_i| + b \qquad (x \in \mathbb{R}).$$

Here the parameter b is a real number, and the natural boundary condition gives us $\sum_{i=1}^{m} a_i = 0$. A unique feature of the univariate case is that we can order the interpolation points $x_1 < x_2 < \cdots < x_m$. Now consider the function s in one of the intervals $[x_i, x_{i+1}]$, $i = 1, \ldots, m-1$. It is clear that in such an interval s is simply a linear function. The demand that s interpolates the data at $x_1, \ldots, x_m$ means that s must be the piecewise linear interpolant to the data in the interval $[x_1, x_m]$. What is the effect of the 'natural' boundary conditions? In the interval $[x_m, \infty)$ we can write

$$s(x) = \sum_{i=1}^{m} a_i(x - x_i) + b = -\sum_{i=1}^{m} a_i x_i + b.$$

Thus s is constant in $[x_m, \infty)$. A similar calculation reveals that s is constant in $(-\infty, x_1]$. Combining all these observations shows that s is the natural linear spline interpolant to the data at $x_1, \ldots, x_m$. This goes some way to explaining why the word 'natural' is appended the boundary or extra conditions. But we can go a little further. It is well known that the natural splines satisfy a variational principle. For the linear spline, if we examine

$$\mathcal{X} = \{f \in \mathcal{S}' : f' \in L^2(\mathbb{R})\},$$

then

$$\int_{-\infty}^{\infty} (s')^2 \leq \int_{-\infty}^{\infty} (f')^2$$

for all $f \in \mathcal{X}$ which also interpolate the data. This variational principle is very useful in developing error estimates, and we shall return to this general thread of ideas later in this account. However, we ought to observe that $\mathcal{S}'$ is the space of tempered distributions, and that the first derivative is to be taken in the distributional sense. There are ways of getting round this distributional approach (see Cheney and Light [10] for an example which corresponds closely to the discussion here), but it does give the most succinct description, and creates the technical background which will underpin all the theory which has been developed in this area. Notice also that the quantity being minimised can be used to specify a seminorm on $\mathcal{X}$ simply by taking the square root of the integral. This seminorm has as kernel $\pi_0(\mathbb{R})$, which is precisely the polynomial subspace we use to augment the original radial basic function. Something very fundamental is happening here. Most mathematicians would regard this seminorm as being a measure of smoothness of the corresponding function. The natural linear spline therefore interpolates the data, and is the smoothest interpolant to the data from $\mathcal{X}$ in the sense that it possesses the smallest derivative in the L^2-norm. If we are to pursue this very natural idea of making higher derivatives of s small, then we will naturally develop seminorms with polynomial kernels. This goes a long way towards explaining the need for augmentation.

Finally in this introduction, we want to discuss briefly the uses to which radial basic

function interpolation is put. There are two significant feelings about interpolation by these functions. Firstly, it is thought that radial basic function interpolation is very good for treating scattered data. Loosely speaking, data is scattered when there is no possibility of determining either a natural choice of coordinate axes, or an origin. It is at the opposite end of the spectrum to gridded data. In the presence of a cartesian product for the data sites, it is much more efficient to use univariate methods together with tensor product constructions to do the interpolation. Secondly, radial basic function interpolation is thought to be very good for dealing with high dimensional data. There is some evidence from the realm of neural networks that this is indeed the case, but we will not venture into the area of high dimensional data interpolation in this paper. Finally, many of the data sets we want to treat have very large numbers of data sites and so our aim is to develop methods which will handle 10,000 to 1,000,000 data sites or more.

2 Computational difficulties and fast evaluation

In this Section, we want to discuss the difficulties that arise when a large radial basic function interpolation problem is posed. We shall also deal with one of the essential tools for overcoming some of the difficulties. The system we want to solve has the form

$$d_j = s(x_j) = \sum_{i=1}^{m} a_i \phi(|x_j - x_i|) + p(x_j) \qquad (j = 1, \ldots, m) \tag{2.1}$$

$$0 = \sum_{i=1}^{m} a_i q(x_i), \qquad \text{for all } q \in \pi_{k-1}(\mathbb{R}^n). \tag{2.2}$$

If we declare a basis for $\pi_{k-1}(\mathbb{R}^n)$ then we can write these equations in matrix form as

$$\begin{pmatrix} A & Q \\ Q^T & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}.$$

Here the matrix A has entries $\phi(|x_j - x_i|)$ and is $m \times m$. The matrix Q has entries $p_\ell(x_j)$, where $p_1, \ldots, p_\nu$ is a basis for $\pi_{k-1}(\mathbb{R}^n)$, and is of size $m \times \nu$. Recall from our assumptions that only low degree polynomials are used, and so Q is a long thin matrix. In the case of thin-plate splines in $\mathbb{R}^2$ it would have size $m \times 3$. However, A is a very large matrix, with absolutely no sparsity. In fact, for thin-plate splines, the matrix A is zero on the diagonal, and has large off-diagonal entries. In solving a large system of linear equations, the only effective strategy is to use an iterative solver. Such a solver will involve many multiplications of the matrix A with a vector a, and the full nature of A makes this a very costly process. One of the key discoveries in this area was the Beatson and Newsam [8] result which showed how fast multipole algorithms could be applied to this area. If we consider the expression

$$s(x_j) = \sum_{i=1}^{m} a_i |x_j - x_i|^2 \ln(|x_j - x_i|) + p(x_j)$$

for some $x_j \in \mathbb{R}^n$, then this can be considered as an evaluation of the function s at the point x_j, or the formation of an element in the matrix vector product Aa. Because of

this, most authors tend only to consider how to *evaluate* the function s in an efficient way — generating what are known as fast evaluation algorithms. It is impossible to estimate properly the importance of this discovery. Anyone involved in programming iterative solutions to the thin-plate spline equations with tens of thousands of points would find that any such algorithm would just grind itself into the dust without this technology. The technology really has two aspects: a mathematical tool, and a programming structure. Here we intend to give only the flavour of the argument. The reader who really wants to know the details is advised to look either at the original paper [8], or the later paper of Beatson and Light [5] which deals with polyharmonic splines. She can also look at two papers which give clear explanations of simple cases. The first is found in a survey paper by Beatson and Greengard [3]. The second is a technical report by Beatson, Levesley and Light [7]. This last paper discusses fast evaluation methods on the circle and higher dimensional spheres, and the reader will find a very careful and full account of the one-dimensional circle case. The first trick with problems in $\mathbb{R}^2$ is to consider complex variables, rather than points in $\mathbb{R}^2$. Let z be a point at which we wish to evaluate s, and u a data point, or *centre*. Then

$$|z-u|^2 \ln|z-u| = \mathcal{RE}(|z-u|^2 \ln(z-u)) = \mathcal{RE}(|z-u|^2 \ln z) + \mathcal{RE}\Big(|z-u|^2 \ln\Big(1-\frac{u}{z}\Big)\Big).$$

Look at the last two expressions here. The first of them has the centre u in the square of the modulus term, and this expression is quite cheap to evaluate, even if there are many centres u. The effect of many centres on the second term is however quite profound, and it is with this term that we must work. The idea is to set a tolerance, and only aim to evaluate s to within this tolerance, rather than exactly. The appropriate series expansion can then be used:

$$|z-u|^2 \ln\Big(1-\frac{u}{z}\Big) = \sum_{p=1}^{\infty} e_p \Big(\frac{u}{z}\Big)^p \approx \sum_{p=1}^{N} e_p \Big(\frac{u}{z}\Big)^p = \sum_{p=1}^{N} f_p(u) z^{-p}.$$

The value of N depends on the tolerance demanded of the evaluation and the relative sizes of u and z. For this reason, we think of z as far away from the origin in $\mathbb{R}^2$, and u close to the origin. If there are now many centres $u_1, \ldots, u_m$ near the origin, and z is far away from the origin, then we can *summarise* the effects of linear combinations of all these centres as follows:

$$\begin{aligned} \sum_{i=1}^{m} a_i |z-u_i|^2 \ln\Big(1-\frac{u_i}{z}\Big) &\approx \sum_{i=1}^{m} a_i \sum_{p=1}^{N} f_p(u_i) z^{-p} \\ &= \sum_{p=1}^{N} \sum_{i=1}^{m} a_i f_p(u_i) z^{-p} = \sum_{p=1}^{N} g_p(u_1, \ldots, u_m) z^{-p}. \end{aligned}$$

The principle now is to use the last expression above to make an approximate evaluation of s. Of course, the assumption that z was far from the origin and $u_1, \ldots, u_m$ were close to the origin is not important. It is simply important that z be far away from the cluster of centres $u_1, \ldots, u_m$. The summarising expression is referred to as a Laurent type expansion, because it summarises the contribution of the centres $u_1, \ldots, u_m$ in terms of

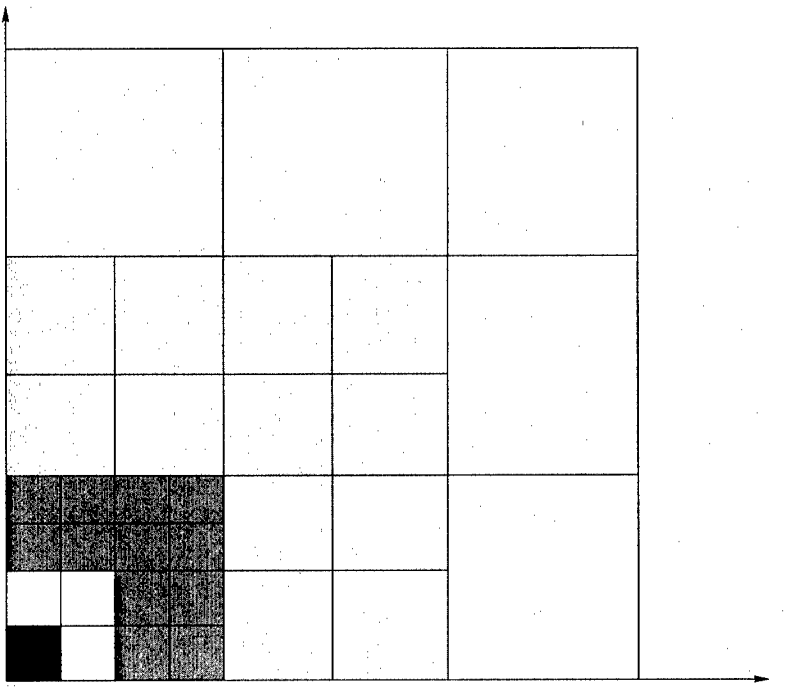

FIG. 1. Fast evaluation panelling.

series involving negative powers of z. There is now a lot of preprocessing to go on before the fast evaluation algorithm is ready to roll. Figure 1 shows how the algorithm proceeds. The shaded square at the bottom left of the domain is the point which contains z, the evaluation point. All the squares around this one which are the same size are deemed to be 'close' to the evaluation square. All other squares are 'far away'. Of course, as the squares get further away from z it becomes possible to use our summarising technique to total up the contributions of larger and larger numbers of points. This is done in a very explicit manner, which is represented by the shading in Figure 1. As we get further away, we double the size of the squares over which we summarise, and there is a band of same-size squares (or a ring, if the evaluation square was in the middle of the domain) two squares wide surrounding the evaluation square. Once all the preprocessing is done, and we shall discuss this a little more in a moment, all the needed coefficients g_p are available, and evaluation can be carried out in about $\mathcal{O}(\log m)$ FLOPS instead of $\mathcal{O}(m)$.

The above account does not quite reveal the whole story. The coefficients g_p are calculated in an orderly manner which greatly improves the efficiency of the algorithm. Suppose our problem is located in $[0,1]^2$. An initial decision is made to divide the original domain into squares of size 2^{-n}. There is then a parent child relationship derived through a quad-tree data structure. The parent $[0,1]^2$ has four children: $[0,0.5]^2$, $[0.5,1]^2$, $[0,0.5] \times [0.5,1]$ and $[0.5,1] \times [0,0.5]$. This parent-child relationship helps in setting up the coefficients $g_p(u_1,\ldots,u_m)$ in an efficient way. There is also a further idea involving

Taylor series, which gives more efficiency. We omit any description of this technique.

3 Inverting the interpolation matrix

Recall as at the beginning of the previous section that the equations specifiying the interpolation problem are as follows:

$$d_j = s(x_j) = \sum_{i=1}^{m} a_i \phi(|x_j - x_i|) + p(x_j), \qquad (j = 1, \ldots, m) \tag{3.1}$$

$$0 = \sum_{i=1}^{m} a_i q(x_i), \qquad \text{for all } q \in \pi_{k-1}(\mathbb{R}^n). \tag{3.2}$$

In matrix terms we have

$$\begin{pmatrix} A & Q \\ Q^T & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix},$$

where A is a full matrix which tends to exhibit poor conditioning. The poor conditioning of A is similar to problems experienced by researchers in the theory of finite elements — as the interpolation points become very dense in a given region, the conditioning gets worse. In fact, there are formal statements relating some impression of the condition number of A (usually the smallest eigenvalue of A) to the minimum interpoint distance. The following table represents the condition number of A when the interpolation points are given on a uniform 5×5 grid in $[0, \alpha]^2$. Of course, on a philosophical level, it does not

Scale parameter α	Condition Number
1.0	3.6458×10^2
0.1	2.5179×10^4
0.01	2.4364×10^6
0.001	2.4349×10^8

TAB. 1 Two norm condition numbers of A.

make any sense whatsoever to describe an interpolation problem as being ill-conditioned. Let's discuss this point in a little more depth. Suppose $x_1, \ldots, x_m$ are points in $\mathbb{R}^n$, and $G_1, \ldots, G_m$ are a set of functions from $\mathbb{R}^n$ to $\mathbb{R}$ which are linearly independent over $\{x_1, \ldots, x_m\}$. That is, interpolation to arbitrary data at $x_1, \ldots, x_m$ by linear combinations of $G_1, \ldots, G_m$ is always uniquely possible. Then there is a basis $F_1, \ldots, F_m$ for the linear span of $G_1, \ldots, G_m$ such that $F_i(x_j)$ is 1 if $i = j$ and is zero for all other values of i, j between 1 and m. If the given data is $d_1, \ldots, d_m$, then the interpolant can be written down immediately as

$$\sum_{i=1}^{m} d_i F_i(x) \qquad (x \in \mathbb{R}^n).$$

If one has in one's hands the basis $\{F_1, \ldots, F_m\}$ and wants to know the coefficients which must be used then one need only invert the identity matrix to obtain the solution, and there are not many matrices which are better conditioned than the identity matrix! Of course, getting one's hands on the basis $F_1, \ldots, F_m$ is usually rather difficult — as hard as solving the original problem in fact. It has become traditional to refer to the basis $F_1, \ldots, F_m$ as the Lagrange basis (in sympathy with the fact that Lagrange was a person who wrote down this basis for polynomial interpolation in one dimension) or the cardinal basis. This last term seems to the author to be quite appropriate, indicating that the basis is special. However, it does not find favour with spline theorists, since they think of the word cardinal in a very technical sense (the interpolation points are $\mathbb{Z}^n$). Terminology aside, the point is still made that the conditioning of any interpolation problem is a function of the available basis. A more practical case of this phenomenon is the problem of natural cubic splines in $\mathbb{R}$. They fit into the radial basic function interpolation scenario, because a natural cubic spline with knots at $x_1, \ldots, x_m$ can be written as

$$s(x) = \sum_{i=1}^{m} a_i |x - x_i|^3 + ax + b \qquad (x \in \mathbb{R}).$$

If we require this spline to interpolate data $d_1, \ldots, d_m$ at $x_1, \ldots, x_m$ then we have to require that $s(x_j) = d_j$ for $j = 1, \ldots, m$. The natural property comes, as expected, from the natural boundary conditions:

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} a_i x_i = 0.$$

The ill-conditioning illustrated in Table 1 would be equally present in this example, and the remark that the conditioning increases as the interpoint spacing decreases would also hold good. Of course, to suggest the use of this basis to a spline practitioner would not be a good idea! We are well used to the idea that B-splines are the correct basis to use in this situation.

I suppose the two principles to emerge from the above discussion are that the basis we have used thus far to describe the interpolation problem is not satisfactory from a computational point of view, and that in at least some of the cases under discussion (all of them one-dimensional) there are other bases which are superior. There are other ways to conceptualise the difficulties we experience with the radial basic functions. Most of them tend to grow at infinity, and have small value at zero. As a general principle, we would like a basis to mimic the B-spline basis. That is, we would like the basis to be local if possible — each basis function having a fairly small support around one of the interpolation points. The first people to make progress in this area were Dyn and Levin [11] in 1983. There is a later paper with Rippa [12] in 1986 which is also worth looking at. Their technique was based on the observation that if $F(x) = |x|^2 \ln |x|$, and $x \in \mathbb{R}^2$, then $\nabla^4 F = 8\pi\delta$. Here, ∇^4 represents the bilaplacian, and δ is the Dirac delta distribution whose action on each rapidly decreasing function in $\mathcal{S}$ is to evaluate it at zero. This description alone should alert us to the fact that $\nabla^4 F = 8\pi\delta$ is a distributional equation, and as such must be handled with care. However, numerical analysts dash in

where others fear to tread, and we can approximate the Laplacian as follows:

$$(\nabla^2 F)(x) \approx h^{-2}\{F(x-he_1)+F(x+he_1)+F(x-he_2)+F(x+he_2)-4F(x)\} \qquad (x \in \mathbb{R}^2).$$

Here h is a real parameter, and e_1 and e_2 are the usual unit vectors in $\mathbb{R}^2$. Pictorially, we can represent this approximation by the stencil shown in Figure 2. The bilaplacian stencil

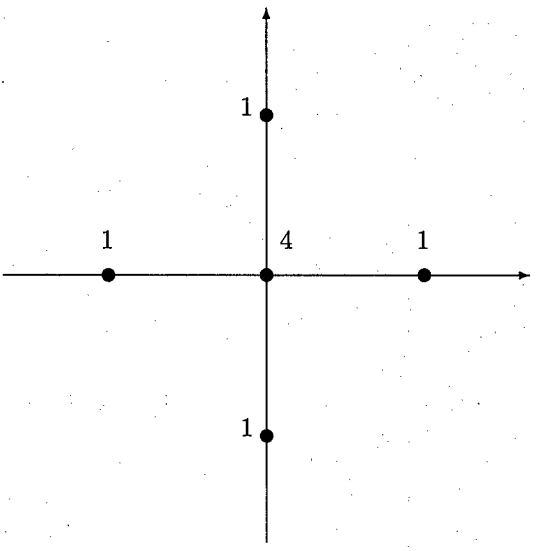

FIG. 2. The stencil for the Laplacian.

is shown in Figure 3. This observation is used in a straightforward way if the interpolation points lie on a grid. Instead of using the thin-plate spline radial basic function to generate a basis, one uses the appropriate linear combinations which represent the bilaplacian of this function. Because one has a distributional equation relating this quantity to the δ function, one does not expect to get the δ function exactly, but one certainly does expect to get a function which decays rapidly at ∞, and this is exactly what happens. Dyn and Levin provide some encouraging numerical results. Of course, there remains the problem of what to do when the data is not gridded. Here one must develop first the appropriate stencil for the Laplacian on a point by point basis. This may seem laborious, but in fact the next few methods we will describe all compute better basis elements on a point by point basis.

Perhaps the most successful class of schemes of this nature — computing a new basis on a point by point approach — comes from Beatson, Goodsell and Powell [2] and Beatson, Cherie and Mouat [1]. Their approach is perhaps simpler to appreciate and implement than that of Dyn and Levin. They begin with the observations I made earlier — what we are really after is the cardinal basis $F_1, \ldots, F_m$ with the property that $F_i(x_j)$ is 1 if $i = j$ and is zero for all other values of i, j between 1 and m. However, because this problem is as difficult to solve as the original one, we proceed as follows. Consider

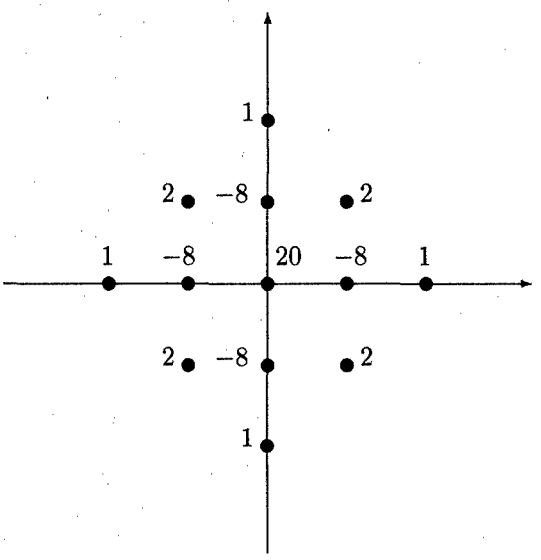

FIG. 3. The stencil for the bilaplacian.

the job of trying to construct F_i. This function is supposed to be 1 at x_i and zero at all other points. Choose about 50 near neighbours of x_i, say $y_j \in \{x_1, \ldots, x_m\}$. This choice must include x_i. Then take

$$F_i(x) = \sum_{j=1}^{50} a_j |x - y_j|^2 \ln |x - y_j| + bx + c, \qquad (x \in \mathbb{R}^2).$$

We demand that

$$F_i(y_j) = \begin{cases} 1 & \text{if } y_j = x_i, \\ 0 & \text{otherwise,} \end{cases}$$

and that the natural boundary conditions are also satisfied. Thus we are producing *approximate cardinal functions* which have the value 1 at the required point, but are only zero on about 50 neighbouring points. This suggestion is based on the fact, observed by many workers, that such functions are often small elsewhere in the domain. We produce some pictures to illustrate this. In the first (Figure 4), 289 points are spaced on a regular grid in $[0, 1]^2$. The approximate cardinal function is based on the 13 points shown in bold in Figure 4. Figure 5 illustrates the same situation, but now as shown the points used to develop the cardinal function are all clustered in one corner of the domain. The effect is to produce significant values at the opposite corner of the domain. One can infer from this that whenever the data is pretty much uniformly distributed, the cardinal functions using points well inside the domain will have good properties, while those at the edge will be poor. Similarly, in a non-uniform distribution, those interior to a cloud of points will behave well, while those at cloud boundaries might not.

There are two methods for dealing with the difficulties which have shown up above.

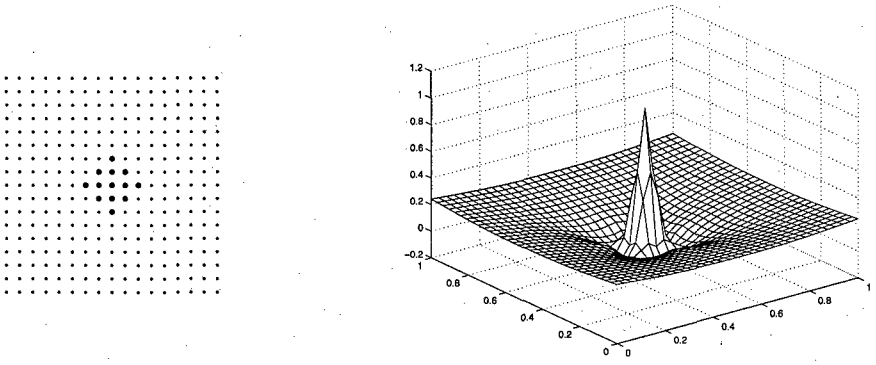

FIG. 4. Approximate cardinal function with points central to the domain.

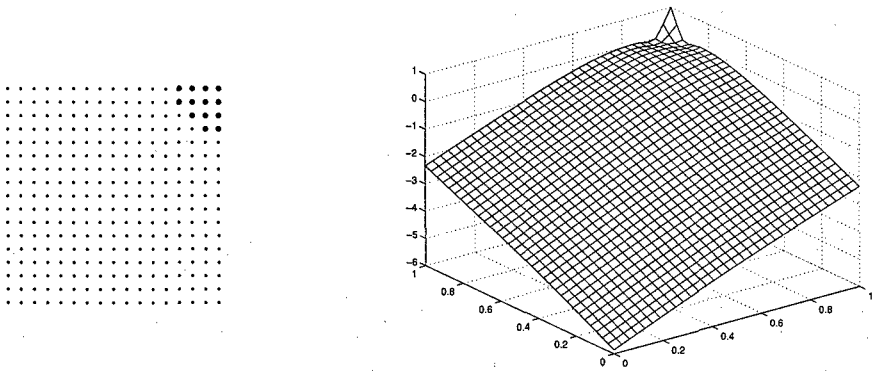

FIG. 5. Approximate cardinal function with points at one corner of the domain.

Firstly, one can *pin* all the cardinal functions at a fixed set of judiciously chosen points — so that every cardinal function must have the value zero at these points. This is very effective in the case of regularly spaced data as Figures 6 and 7 show. One can imagine however, that a data set with a number of clouds might benefit from a judicious choice of points at which to carry out the pinning. What one would really like is a method which does not rely on any user intelligence in the choice of points. As mentioned before, a desirable feature of a good basis function is one which decays at infinity. This decay should be at some rate if possible. The Beatson, Cherie and Mouat prescription for thin-plate splines in $\mathbb{R}^2$ is that the elements should decay like $|x|^{-3}$ as $|x| \to \infty$. There is a problem here, in that if we opt for decay elements everywhere, then we will not obtain a basis for our space. To get around this problem, we accept an element F_i as a decay element if it satisfies

$$\sum_{i=1}^{50} |F_i(y_j) - \delta_{ij}| < \mu$$

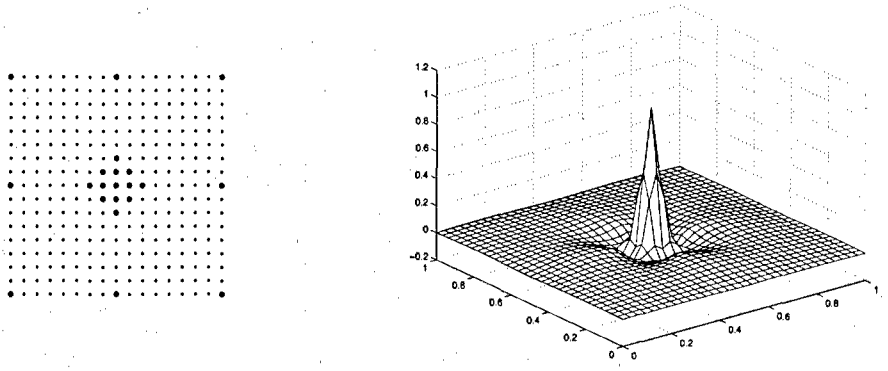

FIG. 6. Approximate pinned cardinal function with points central to the domain.

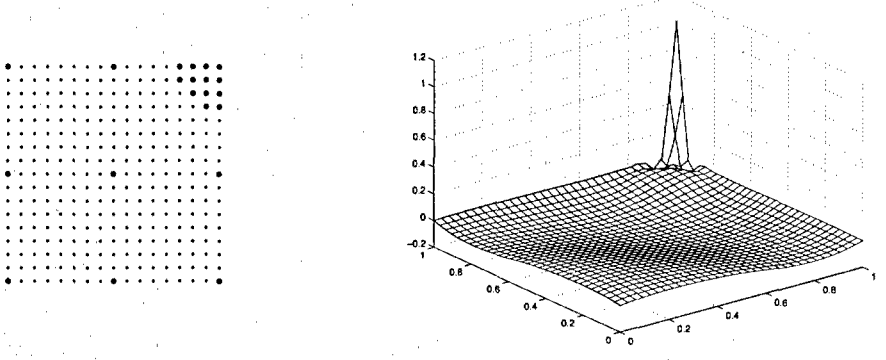

FIG. 7. Approximate pinned cardinal function with points at one corner of the domain.

and

$$|F_i(x)| = \mathcal{O}(|x|^{-3}) \qquad \text{as } |x| \to \infty.$$

Otherwise, we use the F_i which is defined by the previous conditions of cardinality. Again there are a few bells and whistles needed to make this method operate efficiently, but we hope that sufficient detail is present for the reader to be able to see the general idea. All the above methods are providing ways of constructing a better conditioned basis with which to solve the problem. A method still has to be selected to invert the matrix associated with the new basis, which is now much better conditioned than the original matrix corresponding to the conventional basis. The method of choice for most authors is some version of GMRES.

Beatson called the points at which decay could be obtained 'good' points, and points at which decay could not be obtained 'bad' points. This idea has been built on in a recent technical report by Beatson and Levesley [4]. The general spirit is to define good and bad points in the same way as Beatson, and then to develop an iterative solver, solving first on the good, then the bad, then returning to the good and so on.

Finally, a very successful method has recently emerged from the researches of Beatson, Light and Billings [6]. This method has the advantage that it is a fast iterative solver which may be regarded as a preconditioner in its own right (thus it may be combined with a solver such as GMRES). We will describe it here as a solver. It is essentially the domain decomposition method, although as with previous solvers, our description will be very much at a 'bare bones' level, and the interested reader is referred to [6] for the fine details, which include some error estimates, some interesting comments on an alternative basis, and a good deal of theory. We shall describe the method as applied to data on the unit square $[0,1]^2$ in $\mathbb{R}^2$, and we will not make any attempt to make the method adaptive in character. The reader will be able to see these improvements for herself. We will test our method on randomly chosen data in $[0,1]^2$.

We begin with a set of nodes $X = \{x_1, \ldots, x_m\}$ at which interpolation is to be carried out. We will describe the algorithm as it is implemented for solving the thin-plate spline interpolation problem on the node set X. We divide up the square $[0,1]^2$ into a fairly large number of sub-domains $X_1, \ldots, X_\ell$. There are two constraints on these subdomains. It is important that they are constructed so that about equal numbers of points lie in each subdomain — about 50 points per subdomain is ideal. Secondly, it is essential that each subdomain overlap all surrounding subdomains. In our terminology, two subdomains overlap if they have a (small) number of points in common. In each subdomain there are some points in X which lie only in that subdomain and not in any other. We call these points the *inner* points of the subdomain. A *coarse* set Y of inner points in the node set X is also chosen. We will say more about this coarse set in a moment, but at this stage it simply consists of a small number of inner points from each subdomain. The algorithm will then construct the interpolant s and proceeds as follows. We initialise the interpolant s as $s = 0$. We want to solve the equations

$$d_j = s(x_j) = \sum_{i=1}^{m} a_i |x_j - x_i|^2 \ln |x_j - x_i| + \alpha x_j + \beta, \qquad (j = 1, \ldots, m) \tag{3.3}$$

subject to the boundary conditions

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} a_i s_i = \sum_{i=1}^{m} a_i t_i = 0, \tag{3.4}$$

where $x_i = (s_i, t_i)$. In matrix form these equations are

$$\begin{pmatrix} A & Q \\ Q^T & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix},$$

as we have already seen. Our method will operate by residual correction, so we begin by setting

$$r = \begin{pmatrix} d \\ 0 \end{pmatrix}.$$

It is important to recall that α is a vector of length 2, which we write as $\alpha = (\alpha_1, \alpha_2)$. Suppose now we have begun our iterative procedure and generated an approximation s with a residual r. The next few steps describe how to update the approximation and the

residual.

Step 1. We construct $s_1, \ldots, s_\ell$ such that each s_j is an interpolant based only on all points of the subdomain X_j, using as data the residual vector r restricted to X_j.

Step 2. For each inner point x we now have a single real number a_x which is the coefficient of $|\cdot - x|^2 \ln|\cdot - x|$. If we look at the collection of coefficients belonging to all the inner points of all domains, then this collection is not in general orthogonal to π_1. That is, they fail to satisfy boundary conditions of the type given in Equation (3.4). We now correct so that the collection of coefficients corresponding to all inner points of all domains is orthogonal to π_1.

Step 3. We set

$$\mathcal{S}_1 = \sum \{a_x |\cdot - x|^2 \ln|\cdot - x| : x \text{ is an inner point}\}. \tag{3.5}$$

Step 4. We evaluate the residual $\mathcal{R} = r - \mathcal{S}_1$ at the coarse grid points, and then construct the interpolant $\mathcal{S}_2$ to this residual on the coarse grid points Y.

Step 5. We update s by $s \leftarrow s + \mathcal{S}_1 + \mathcal{S}_2$. The new residual is then given by

$$r = \begin{pmatrix} z \\ 0 \end{pmatrix},$$

where

$$z_i = d_i - s(x_i), \qquad i = 1, \ldots, m.$$

This iterative process can either be continued to convergence, or used as a preconditioner followed by GMRES. Table 2 shows some run times taken to obtain an error of less than 1×10^{-6} for the Franke 1 function (see [13] for the definition of this function). Random nodes were generated in $[0,1]^2$ and an Intel Celeron PC was used. Recently,

Number of nodes	Number of iterations	Time (seconds)
10,000	8	7.0
20,000	8	17.5
40,000	6	35.5
80,000	6	105.7
160,000	7	407.8

TAB. 2 Run times for domain decomposition.

the group at Leicester, using a twin processor Compaq PC, has obtained solutions to a problem with 1,000,000 random points in less than 9 minutes, and we can safely say that the combination of domain decomposition methods and multipole fast evaluation has produced a robust and effective method. Most practitioners will be aware of other ways to run a domain decomposition algorithm. In particular, one can use a nesting approach where one starts with only four subdomains each containing large numbers of points. To solve each subdomain problem, one subdivides again and does domain decomposition in the subdomain.

Bibliography

1. Beatson, R.K., J.B. Cherie and C.T. Mouat, *Fast fitting of radial basis functions: methods based on preconditioned GMRES iteration.* Advances in Computational Mathematics, **11** (1999), 253–270.
2. Beatson, R.K., G. Goodsell and M.J.D. Powell, *On multigrid techniques for thin plate spline interpolation in two dimensions*, Lectures in Applied Mathematics **32** (1996), 77–97.
3. Beatson, R.K. and L. Greengard, *A short course on fast multipole methods*, in *Wavelets, multilevel methods and elliptic PDEs*, Ainsworth, M., J. Levesley, W.A. Light and M. Marletta (eds), Oxford University Press, Oxford (1997), 1–38.
4. Beatson, R.K. and J. Levesley, *Good point/bad point iterations for solving the thin-plate spline interpolation equations*, University of Leicester Technical Report, 2001/34 (2001).
5. Beatson, R.K. and W.A. Light, *Fast evaluation of radial basis functions: Methods for two-dimensional polyharmonic splines*, IMA Journal of Numerical Analysis **17** (1997), 343–372.
6. Beatson, R.K., W.A. Light and S. Billings, *Domain decomposition methods for solution of the radial basis function interpolation problem*, SIAM Journal Scient. Stat. Comp. **22**(5) (2001), 1717–1740.
7. Beatson, R.K., J. Levesley and W.A. Light, *Fast evaluation of radial basic functions on spheres*, preprint.
8. Beatson, R.K. and G.N. Newsam, *Fast evaluation of radial basis functions I*, Computers and Mathematics with Applications, **24** (12) (1992), 7–19.
9. Beatson, R.K. and M.J.D. Powell, *An iterative method for thin plate spline interpolation that employs approximations to the Lagrange functions*, in *Numerical Analysis 1993*, D.F. Griffiths and G.A. Watson (eds), Longmans, Harlow, 1994.
10. Cheney, E.W. and W.A. Light, *A course in approximation theory*, Brooks Cole, Pacific Grove Ca, 1999.
11. Dyn, N. and D. Levin, *Iterative solution of systems originating from integral equations and surface interpolation*, SIAM J. Numer. Anal. **20** (1983), 377–390.
12. Dyn, N., D. Levin and S. Rippa, *Numerical procedures for surface fitting of scattered data by radial functions*, SIAM Journal Scient. Comp. **7**
13. Franke, R., *Scattered data interpolation: Tests of some methods*, Mathematics of Computation **38** (1982), 181–200.
14. Mairhuber, J.C., *On Haar's theorem concerning Chebychev approximation problems having unique solution*, Proc. Amer. Math. Soc. **7** (1956), 609–615.
15. Micchelli, C.M., *Interpolation of scattered data: distance matrices and conditionally positive definite functions,* Constr. Approx. **2** (1986), 11–22.
16. Sibson, R. and G. Stone, *Computation of thin-plate splines*, SIAM Journal on Scient. Stat. Comp. **12** (1991), 1304–1313.